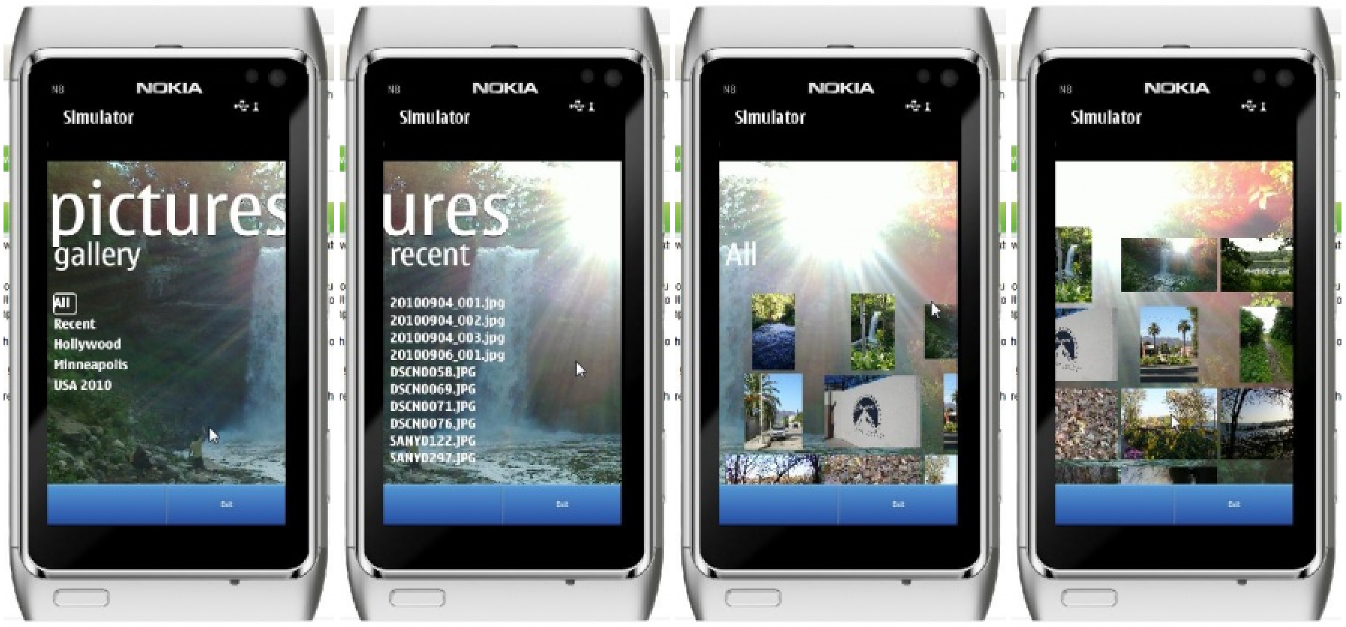
Can QtQuick create a Windows Phone Hub

Love it or hate it, Windows Phone 7 is a unique interface and Windows Phone will be in the future for Nokia smartphones. One of the unique features of the UI is it's Hubs for things like picture, people and music.

I set about trying to recreate this in QtQuick, and although I had hoped by now to have a full working project for this out in the projects area of Forum.Nokia, I haven't got to that point yet. I have a few UI issues to resolve too but have also resolved some things now. Ultimately, this is still a far less impressive Wiki than I hoped for, though it can now begin to be instructive, although I'm still figuring a lot out myself. I will continue to update this as I complete this work.

The answer to the question this wiki poses is increasingly Yes however, though i think for some of the next steps I will need to get to grips with how C++ and QtQuick work together. So this is what I have now:

- A hub with 3 pages that scroll properly, including the wide one with the pictures grid
- A clickable list on the first that updates the content of the others
- A background image that moves as you scroll, but at a different speed to the foreground - unfortunately, QmlViewer wouldn't record video of me of this.
- A heading that also moves as you scroll



So rather than dump my whole code out for everyone, here are some bit for getting the features of hubs working. It's important to note that the source code snippets i this have been trimmed to remove properties that are not necessarily a part of resolving the challenges I worked through. I have also not included any snippets from the ugly JavaScript file I created to load the images and text for this hub.

The shell

```
Rectangle {
    id: main

    //for uniform fonts
    property color textColor: "#ffffff"
    property string fontName: "calibri"

    //the scrolling images
    Image{
        id:backgroundImage
    }

    Component.onCompleted: initialize()
    function initialize() {  ImageLoader.createAlbums();
                             ImageLoader.createRecents(0);
```

```
                                    ImageLoader.createImages(0);

                                }

//List view with VisualItemModel
    VisualItemModel {
        id: itemModel

        //First hub tile - pictures menu
        Rectangle {
            id: rect1
            //the first page also contains the heading
            Text {
                id: headingText
                text: "pictures";
            }
           }

        //Second hub page - the recent items list
        Rectangle {
            id: rect2
            Text { color: textColor;text:  "recent"}
            }


        //Third hub page - the pictures
        Rectangle {
            id: galleryRect
            width: main.width*2.5; height: view.height

            Text {id: galleryText; color:textColor; text: "All";

        }
    }
    //controls everything (a lot anyway)
    ListView {
        id: view
    }
}
```

So we have a simple set up. A ListView and VisualItemModel containing 3 Rectangles - one for each page. Each is a little different that the others - the first contains the hearer text for the hub, the second is pretty standard and the third is 2.5 times wider than the other 2. Aside from the list view, the other visual element is an image, which is the background image that scrolls at a different speed to the list. The final thing to mention is there is a function that launches when the page has loaded, and that calls 3 functions to load the albums, recent file list and images from a separate file.

The master ListView

The first part was getting this list view to work for me. I used the same system described in QML paging using ListView here, but modified with some JavaScript, I was able to make it allow me to drag into the wider hup page.

```
ListView {
        id: view
        opacity: 100
        anchors.fill: parent;
        model: itemModel
        orientation: ListView.Horizontal
        snapMode: ListView.SnapOneItem;
        highlightRangeMode: ListView.StrictlyEnforceRange
        flickDeceleration: 500
        cacheBuffer: 200;
        anchors.bottomMargin: 30
        boundsBehavior: Flickable.DragOverBounds

        //code to track the position of the content and allow me to know when the
current page is the wider one.
        property int previousX;
        onMovementStarted: previousX=contentX
        onContentXChanged:{ if ((view.contentX >= galleryRect.x) && (view.contentX
<=(galleryRect.x+galleryRect.width-30)))
                          {
                              if (previousX>contentX && contentX<(galleryRect.x+50)) {
                                    setSnapMode(true);
                              }else {
                                    setSnapMode(false);
                              }
                          } else {
                             setSnapMode(true);
                         }
                        backgroundImage.x=(0-view.contentX)/2;
                        headingText.x=(0+view.contentX)/2;
        }

        function setSnapMode(on){
            if (on==true){
                view.snapMode=ListView.SnapOneItem;
                view.highlightRangeMode= ListView.StrictlyEnforceRange;
            } else if (on==false) {
                view.snapMode=ListView.NoSnap;
                view.highlightRangeMode= ListView.NoHighlightRange;
            }
        }
    }
```

The ListView controls a lot here. The properties set it up the same as in the QML Paging Using List View wiki, which would be great if all the pages were the same size. To allow us to scroll to the middle of the gallery page we need to turn of SnapMode and the HighlightRangeMode. To do that, a javascript function compares the current contentX position with the x position of our wide page (galleryRect) and if it's greater than it, switches mode by calling setSnapMode(false). As we drag back towards the edge of the galleryRect, we turn it on, so it will snap to the edge of the image gallery, before moving on to the previous pages.

The background Image and Title

The method also has two lines unrelated to this - these shift the x position of the background image and the title so they move as you flick between pages, but they move at a different speed. The background image and Title have no anchors, but use x and y properties instead so that these values can be changed through property binding. I used the PreserveAspectCrop fillMode so that both landscape and portrait images would look OK and not distorted, and also fill the space.

```
 Image{
        height: main.height
        source: ImageLoader.getRandomImage()
        fillMode: Image.PreserveAspectCrop
        y: main.y
        x:0;
        width: (rect1.width+rect2.width+galleryRect.width)/2
        id:backgroundImage
```

```
    }

//List view with VisualItemModel
    VisualItemModel {
        id: itemModel

        //First hub tile - pictures menu
        Rectangle {
            id: rect1
            //the first page also contains the heading
            Text {
                id: headingText
                text: "pictures";
                font { family:fontName; bold:false; pointSize: 38}
                smooth: true
                anchors.leftMargin: 10
                style: Text.Raised
                x:0
                y:0
                opacity:1
                color: "white"
            }
```

The first hub List View - recording list view clicks

Another challenge was getting the ListView used for the album list to respond to mouse/screen clicks - This isn't really explained anywhere in the ListView example documentation. Ultimately, I discovered that a mouseArea was needed on the delegate, not on the ListView, and that the only way to set the currentIndex appeared to be to pass the index of the delegate - I'm surprised that the ListView natively handle's flicking, but not clicking. The "album" property is populated in the ImageLoader JavaScript file

```
//First hub tile - pictures menu
        Rectangle {
            id: rect1

            Text {
                id: headingText
            }
            Text { color: textColor; text:  "gallery";
              }
           Rectangle {
               color: "#00000000"
               //top margin give space for header text
               anchors{ left:parent.left; right:parent.right; bottom:parent.bottom;
top:parent.top
                   leftMargin: 10; rightMargin:10; bottomMargin:10; topMargin:200 }

                   VisualDataModel {
                       id: galleryItemModel
                       model :ListModel {
                           id: albumModel
                       }

                       delegate:
                           Rectangle{
                           width: itemText.width+10; height: itemText.height+5
                           color: "#00000000"
                           id:menuItem
```

```
                            Text{id:itemText
                                    font{family:fontName; bold: true} color: textColor
                                    text: album
                                }
                                MouseArea{
                                    anchors.fill: parent
                                    onClicked: albumList.currentIndex = index
                                 }
                            }
                  }
                   ListView {
                        id: albumList
                        anchors.fill: parent;
                        model: galleryItemModel
                        orientation: ListView.Vertical
                        flickDeceleration: 2000
                        cacheBuffer: 200
                        highlightFollowsCurrentItem: true
                        highlight: Rectangle { width: currentItem.width; color:
"#00000000"; border {color:"#ffffff"; width:2} radius: 5}
                        currentIndex: 0

                        onCurrentIndexChanged: {
                            //allows the border to resize to match the text on each
delgate
                            highlightItem.width=currentItem.width;
                            //loads the recent images list and image gallery for the
selected album, as well as setting the gallery heading
                            ImageLoader.createRecents(currentIndex);
                            ImageLoader.createImages(currentIndex);
                        }
                   }
               }
```

The rest (and code download)

The recent images list and the gallery were fairly standard list views and grid views from what I can tell, so no code examples for those yet. However, there are a few things i need to get working still which affect those parts, and a few which don't.

- Clicking image or recent filename opens the selected image full-screen, with close button to return to hub
- Add C++ code to use actual images from device
- In Windows Phone, the hub's wrap so after the gallery it goes to the album page again - no idea how to do this
- See to what extent this can be made into a component
- Use the component to create something else.

Additionally, I don't have a Windows Phone so I may get the look and feel totally wrong. I will be updating this as I make progress, until this becomes a great how-to with cleaner source code and a project to look at somewhere. In the meantime, if you would like to look at my dirty sourcecode for this, here it is: File:PicturesHubFiles.zip

This file contains a project file, picturesHub.qml, ImageLoader.js and a readme, plus some empty folders for the folder structure for images. There are no images, so please add a few and replace the paths in ImageLoader.js with the filenames for your images.